UNITED STATES PATENT APPLICATION

for

Method to Collect Address Trace of Instructions Executed

Inventors:

Rangarajan R. Calyanakoti

Veerasham Bukka

Manish Singh

Brian Pollard

Prepared by:

BLAKELY SOKOLOFF TAYLOR & ZAFMAN LLP
12400 Wilshire Boulevard
Los Angeles, CA 90025-1030
(408) 720-8300

Attorney's Docket No.: 042390.P16968

"Express Mail" mailing label number: EV339922763US
Date of Deposit: 9/25/03
I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service "Express Mail Post Office to Addressee" service on the date indicated above and that this paper or fee has been addressed to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

JUANITA BRISCOE
(Typed or printed name of person mailing paper or fee)

Juanita Briscoe
(Signature of person mailing paper or fee)

9/25/03
(Date signed)

Method to Collect Address Trace of Instructions Executed

FIELD OF INVENTION

[0001] The present invention relates to computer programming, and more particularly, to collecting addresses of instructions in a computer program.

BACKGROUND

[0002] Typically, to build software executable on a processor, one or more program files written in a high level programming language, such as, C, C++, Java, etc., are compiled to generate one or more intermediate files in object code or assembly languages. The intermediate files may then be optionally linked to form an executable.

[0003] Software executable on a processor is usually loaded in a memory as one or more files containing text and data. The text, which is also referred to as the code part or the code segment, contains the encoded instructions that are executed to perform the business logic of the software. The processor loads successive instructions from the memory and executes them in its execution unit. Loops, branches, and program counter modifying instructions cause changes in the program flow and cause the execution of many instructions more than once and in different orders based on several run-time flow and conditions that the software might experience. To fetch these instructions, the processor reads from the memory at various addresses at which these instructions are stored.

[0004] By studying the address trace of the instructions executed, one can improve the efficiency of various hardware components and software operations, such as, the memory controller hub, memory caches, memory banking, memory paging, etc. In particular, software developers use the instruction addresses to identify the most

frequently used portions of computer programs, which is helpful in improving the efficiency of the programs, memory usage, etc. For example, appropriate portions of the computer programs may also be locked in fast memory or caches to improve the efficiency of program execution. Analysis of the address information also helps one to procure source code coverage of the computer programs.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] The present invention will be understood more fully from the detailed description that follows and from the accompanying drawings, which however, should not be taken to limit the appended claims to the specific embodiments shown, but are for explanation and understanding only.

[0006] Figure 1A shows a flow diagram of an embodiment of a process for generating an output file to collect address trace of instructions executed.

[0007] Figure 1B shows a flow diagram of one embodiment of a process to instrument the software.

[0008] Figure 2A shows a section in a sample software.

[0009] Figure 2B shows the instrumented code generated by an embodiment of a process to instrument the software.

[0010] Figure 3A shows a section in a sample software.

[0011] Figure 3B shows the instrumented code generated by one embodiment of a process to instrument the software.

[0012] Figure 4 shows an exemplary embodiment of a computer system.

DETAILED DESCRIPTION

[0013] A method and an article of manufacture for generating instrumented software to collect addresses of instructions in a software program in the order in which the instructions are executed by a processor is disclosed in the following description. Numerous specific details are set forth in the description. However, it is understood that embodiments of the invention may be practiced without these specific details. In other instances, well-known modules, structures, and techniques have not been shown in detail in order not to obscure the understanding of this description.

[0014] Some portions of the detailed descriptions that follow are presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the tools used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of operations leading to a desired result. The operations are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

[0015] It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussion, it is appreciated that throughout the description, discussions

utilizing terms such as "processing" or "computing" or "calculating" or "determining" or "displaying" or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

[0016] The present invention also relates to apparatus for performing the operations herein. This apparatus may be specially constructed for the required purposes, or it may comprise a general-purpose computer selectively activated or reconfigured by a computer program stored in the computer. Such a computer program may be stored in a computer readable storage medium, such as, but is not limited to, any type of disk including floppy disks, optical disks, CD-ROMs, and magnetic-optical disks, read-only memories (ROMs), random access memories (RAMs), EPROMs, EEPROMs, magnetic or optical cards, or any type of media suitable for storing electronic instructions, and each coupled to a computer system bus.

[0017] The algorithms and displays presented herein are not inherently related to any particular computer or other apparatus. Various general-purpose systems may be used with programs in accordance with the teachings herein, or it may prove convenient to construct more specialized apparatus to perform the required operations. The required structure for a variety of these systems will appear from the description below. In addition, the present invention is not described with reference to any particular programming language. It will be appreciated that a variety of programming languages may be used to implement the teachings as described herein.

[0018] A machine-readable medium includes any mechanism for storing or transmitting information in a form readable by a machine (e.g., a computer). For example, a machine-readable medium includes read only memory ("ROM"); random access memory ("RAM"); magnetic disk storage media; optical storage media; flash memory devices; electrical, optical, acoustical or other form of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.); etc.

[0019] Instrumentation is a technique to alter a software program to achieve certain specific goals. Alteration may be done on software program in a variety of forms, such as, source code files written in high level languages, intermediate files, like assembly or object files, or final executable in binary code. In the following description, instrumentation generally refers to inserting certain instructions in a software program. In one embodiment, the instrumented software executes original instructions in the software and outputs the address trace of the original instructions executed. In one embodiment, an instrumentation source block (ISB) is inserted before an instruction in a program to determine the address of the instruction during execution of the program. The file with the ISB is hereinafter referred to as the instrumented software.

[0020] Figure 1A shows a flow diagram of an embodiment of a process to collect addresses of instructions in a software program executed in time order. The process is performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. The source 101 is in a format, where each instruction is discretely identifiable. The source 101 may be in assembly code, object code, binary code, etc. In contrast, a source that is in a format where the instructions are

non-discretely identifiable, such as source 103, has to be converted into a format where individual instructions are discretely identifiable. In one embodiment, the source 101 is derived from a user-created source 103, in which individual instructions are not discretely identifiable. The user-created source 103 may be coded in a high level programming language, such as, C, C++, Java, etc. In one embodiment, the user-created source 103 is input to the processing block 105 to generate the source 101. The processing block 105 may include a compiler or a linker to generate the source 101. However, one should appreciate that the source 101 may be generated in other manners.

[0021] Processing logic receives the source 101 (processing block 110) and performs instrumentation on the source 101 (processing block 160) to generate an instrumented software (data block 165). More details of instrumentation are discussed below. Processing logic executes the instrumented software (processing block 170) to generate the output 172 originally intended by the user-created source (source 101 or source 103) and the address trace 174 of the instructions in the source 101.

[0022] Figure 1B shows a flow diagram of an embodiment of a process to instrument software. The instrumented software when executed would output the addresses of the instructions in the order in which they are executed. The process is performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. The individual instructions in the source 101 are discretely identifiable. The source 101 may include assembly code, object code, binary code, etc. In one embodiment, the source 101 is derived from a user-created source 103, which is in a high level programming language, such as, C, C++, Java, etc. In one

embodiment, the user-created source 103 is input to the processing block 105 to generate the source 101. The processing block 105 may include a compiler or a linker to generate the source 101. However, one should appreciate that the source 101 may be generated in other manners.

[0023] Processing logic receives the source 101 of a software program (processing block 110). In one embodiment, processing logic initializes an index, n (processing block 115). Processing logic then retrieves instruction n from the source 101 (processing block 120). Processing logic determines an offset caused by the additional instructions inserted so far for instrumentation of the software program, i.e., until and including the instructions added by inserting the *n*th ISB (processing block 125). Processing logic then inserts the instructions for storing the context (Backup) (processing block 127), the ISB (processing block 130), and the instructions for restoring the context (Restore) (processing block 133). The determination and the use of the offset will be further discussed in the examples below. More details on the context, as well as storing and restoring the context are described in a later section. Processing logic then inserts the instruction n in the instrumented software (processing block 135). In one embodiment, processing logic increments n (processing block 140) and checks whether there are any more instructions in the source 101 (processing block 145). If there is, processing logic repeats the processing blocks 120-145 in one embodiment. Otherwise, processing ends (processing block 150).

[0024] In one embodiment, the ISB includes instructions to determine the address of the corresponding instruction executed after the ISB. In one embodiment, the ISB

includes instructions that branch to another module or library that contains instructions to determine instruction address.

[0025] In one embodiment, an individual inserts the ISB into the source 101. In an alternate embodiment, the ISB is inserted automatically into the source 101 by a tool, such as, for example, a script. In one embodiment, the instrumented software may be further modified. In one embodiment, an ISB is inserted before every instruction. In one embodiment, an ISB is inserted before only some of the instructions. In one embodiment, instructions that have to be executed consecutively are identified and no ISB is inserted in between these instructions. Furthermore, when an ISB cannot be inserted amidst consecutive lines of code, the size of the ISB may differ from other ISBs inserted in other parts of the source 101. However, one should appreciate that the exact size of each ISB and the instructions within the ISB are machine specific. In one embodiment, the ISB outputs the address of the instruction following the ISB during the execution of the instrumented software. In one embodiment, the processor executing the instrumented software includes one or more control registers, such as, for example, the program counter or the link register. The control registers contain the addresses of the instructions to be executed next. In one embodiment, the ISB determines the address of the following instruction using the contents of the control registers.

[0026] In one embodiment, an individual codes the ISB in a format where individual instructions are discretely identifiable, such as, for example, in binary code or assembly code. In an alternate embodiment, an individual first codes the ISB in a high level programming language, such as, C, C++, Java, etc., and then the ISB is converted into another format, such as, assembly code or binary object code, using a tool, such as a

compiler, so that each instruction is discretely identifiable. A user may further modify the resulting ISB as appropriate. The exact instructions in the ISB are dependent on the processor that executes the program because each processor may have its own instruction set.

[0027] In one embodiment, the processor fetches successive instructions from the memory and executes the instructions fetched in an execution unit of the processor. In one embodiment, the inserted ISB is executed to output the address or addresses of an instruction or instructions executed for which this ISB is inserted. Certain instructions may cause a break in the sequence of the flow of instructions, such as, for example, branch instructions, instructions that modify the value of the program counter, etc. An ISB placed after such an instruction will not necessarily be executed even when the instruction has been executed because the program flow may jump to an instruction other than the ISB following the instruction. Therefore, the ISB for determining the address of an instruction is placed before the corresponding instruction to ensure that the ISB is executed whenever the instruction is executed.

[0028] One embodiment of a processor maintains run-time status information affecting the execution of the instructions in the program. In one embodiment, the information is maintained in various control and status registers. In one embodiment, the processor provides a set of registers for manipulating program data. The set of registers may include a number of general-purpose registers (GPRs). An embodiment of the ISB may use one or more of the GPRs. The control and status information in the registers are hereinafter referred as the context. In one embodiment, the inserted ISB alters the current context of the software executing on the processor, and hence, affects the execution of the

software. Therefore, in one embodiment, additional instructions are inserted to store the context before an ISB and to restore the context after the ISB. The example shown in Figures 2A and 2B illustrates the concept discussed above.

[0029] Referring to Figure 2A, a section of an example of the source is shown. The section includes instructions n through n+2. The amount of instrumentation, i.e., the number of instructions inserted during instrumentation, before instruction n is i_n. After performing one embodiment of the process to instrument the source in Figure 2A, the instrumented software in Figure 2B is resulted. The instructions to store the context are represented as "*Backup Context*" in Figure 2B, and likewise, the instructions to restore the context are represented as "*Restore Context*" in Figure 2B. Referring to Figure 2B, a "*Backup Context*," an ISB, and a "*Restore Context*" are inserted before each of instructions n through n+2. For example, the instructions 205 are inserted before instruction n 210 in Figure 2B. Furthermore, the instrumentation may cause a skew in the instruction address. The skew is discussed in details below. In one embodiment, the amount of instrumentation inserted before each instruction is provided to the corresponding ISB as an offset for determining the non-skewed address of the instruction.

[0030] In one embodiment, the instruction addresses of the instrumented software are skewed because the instructions for storing the context, the ISB, and the instructions for restoring the context themselves add one or more instructions into the instrumented software. The instructions added skew the addresses of the original instructions in the instrumented source. The skew may be removed by providing each ISB with the amount of instrumentation done prior to the corresponding instruction in the source. In one embodiment, the offset includes both the skew introduced by the ISB and the associated

instructions that backup and restore the context. Using the offset, one embodiment of the ISB calculates the address of the corresponding original instruction. Hence, the resulting execution of the instrumented software would output the instruction addresses without skew. The technique of skew removal is further illustrated by the example shown in Figures 3A and 3B.

[0031] Figure 3A shows an example of a source 310. In one embodiment, the instrumented software 320 in Figure 3B is generated from the source 310 in Figure 3A by the process illustrated in Figure 1B. The hexadecimal number in front of each instruction is the address of the instruction. Each instruction in Figures 3A and 3B has a size of 4 bytes in one embodiment. Instructions of other sizes, such as, for example, 8 bytes, 16 bytes, etc., may be used in other embodiments. For the purpose of illustration, only two of the instructions in the source 310 are shown in Figure 3A, namely, 318 and 319. However, it should be apparent to one of ordinary skill in the art that a source may include various numbers of instructions. Furthermore, one should appreciate that the type of instructions shown in Figure 3A is for illustration only and the appended claims are not limited to any particular type of instructions. The instructions 318 and 319 in Figure 3A correspond to the instructions 328 and 329 in Figure 3B respectively. For each of the instructions 318 and 319, instructions to store the context, an ISB, and instructions to restore the context are inserted before the instruction in the instrumented software 320. For example, the instructions 321 to store the context, the ISB 323, and the instructions 325 to restore the context are inserted before the original instruction 328 in the instrumented software 320.

[0032] Referring to Figure 3B, the ISB includes a "branch-to-routine print-to-console" instruction to cause the processor to jump to the "print-to-console" routine to print the address of each of the corresponding original instructions 328 and 329 onto the console. In an alternate embodiment, another routine or section of code is used to make the instrumented software to output the address of the original instruction to another medium, such as, for example, a text document. In one embodiment, the text document is sent to a printer to be printed. In one embodiment, the addresses of the original instructions are output into a file to be further analyzed by a processing logic, which may include software, hardware, or a combination of both. The additional instructions of the routine print-to-console may be in a binary object file or a binary library. However, one may take into account of the skew caused by the additional instructions of the routine print-to-console. In one embodiment, the instructions from the binary object file or the binary library are added at the end of the list of program binary files or assembly files provided to the linker in order to avoid introducing a skew to the address of the original instructions in the program. In an alternate embodiment, the instructions to implement routines, such as, print-to-console, are embedded in the ISB itself, and thus, obviating the need for an additional library or binary file.

[0033] In addition to the "branch-to-routine print-to-console" instruction, the embodiment of the ISB in Figure 3B includes a "mov r0" instruction to provide an offset (or a parameter in terms of software programming terminology) to the routine, "print-to-console." As discussed above, the ISB and the instructions to store and to restore the context introduce a skew in the address of the original instruction in the instrumented software 320. The "mov r0" instruction puts an offset corresponding to the amount of

instrumentation inserted so far into register r0. In one embodiment, the offset includes the number of the instructions added to store the context, the ISB, as well as the instructions to restore the context that have been inserted so far. For example, referring to Figure 3B, there are 6 instructions added before each original instruction. Since the instruction 318 is the first instruction in the source 310 in Figure 3A, therefore, an offset of 6 instructions * 4 bytes/instruction = 24 bytes is provided to the ISB 323 corresponding to the original instruction 318. Likewise, an offset of 2 * 6 instructions * 4 bytes/instruction = 48 is provided to the corresponding ISB of the second original instruction 319.

[0034] As mentioned above, the instructions 328 and 329 in the instrumented software 320 in Figure 3B correspond respectively to the original instructions 318 and 319 in the source 310 in Figure 3A. One would expect to output the instruction addresses of the instructions 318 and 319, i.e., 0x00008000 and 0x00008004 as the address trace of the instructions of the sample software as shown in Figure 3A. But as can be seen, the addresses of the corresponding instructions 328 and 329 in the instrumented software 320, i.e., 0x00008018 and 0x00008034, are different, and hence, skewed. During instrumentation, ISBs and instructions to store and to restore the context are inserted into the instrumented software, causing skews in the original instructions of the source 310. Thus, the offsets are provided to the corresponding ISB's to remove the skew. To print the addresses of the instructions in the original source 310 in time order, the skews are removed before printing the addresses. To remove the skew, the "print-to-console" routine subtracts the offset from the address of the corresponding original instruction in the instrumented software 320. For example, the address of the instruction 328 in the

instrumented software 320 is 0x00008018. The offset provided to the ISB corresponding to the instruction 328 is 24 (i.e., 0x00000018). Therefore, the address of the instruction 328 after removing the skew is 0x00008018 – 0x00000018 = 0x00008000 which is the address of the corresponding original instruction 318. In one embodiment, the "print-to-console" routine uses one or more registers, such as, for example, the link register, to calculate the address of the corresponding original instruction.

[0035] One should appreciate that the instructions and the assembly code shown in the above examples are for the purpose of illustration only. They should not be construed to limit the appended claims to any particular type of code or any particular processor supporting a certain type of assembly code or object code.

[0036] In one embodiment, the instrumented software is passed through one or more tools, such as, for example, a linker, to be converted into an executable file. In an alternate embodiment, the instrumented software itself is already in a format executable by a processor. In one embodiment, a processor in a computer system runs the executable file. In one embodiment, during execution of the instrumented software, the ISB in the executable file determines and outputs the addresses of the instructions in the source 101 (referring to Figure 1) in the order in which the instructions are executed by the processor.

[0037] In one embodiment, one or more of the operations described above are implemented by a computer system. Figure 4 shows an exemplary embodiment of a computer system. The computer system 400 comprises a communication mechanism or bus 411 for communicating information, and a processor 412 coupled with bus 411 for

processing information. Processor 412 includes a microprocessor, but is not limited to a microprocessor, such as, for example, Pentium®, Itanium®, PowerPC®, etc.

[0038] System 400 further comprises a dynamic random access memory (DRAM), or other dynamic storage device 404 (referred to as main memory) coupled to bus 411 for storing information and instructions to be executed by processor 412. Main memory 404 also may be used for storing temporary variables or other intermediate information during execution of instructions by processor 412.

[0039] Computer system 400 also comprises a read only memory (ROM) and/or other static storage device 406 coupled to bus 411 for storing static information and instructions for processor 412, and a data storage device 407, such as a magnetic disk or optical disk and its corresponding disk drive. Mass storage device 407 is coupled to bus 411 for storing information and instructions.

[0040] Computer system 400 may further be coupled to a display device 421, such as a cathode ray tube (CRT) or liquid crystal display (LCD), coupled to bus 411 for displaying information to a computer user. An alphanumeric input device 422, including alphanumeric and other keys, may also be coupled to bus 411 for communicating information and command selections to processor 412. An additional user input device is cursor control 423, such as a mouse, trackball, track pad, stylus, or cursor direction keys, coupled to bus 411 for communicating direction information and command selections to processor 412, and for controlling cursor movement on display 421.

[0041] Another device that may be coupled to bus 411 is hard copy device 424, which may be used for printing instructions, data, or other information on a medium such as paper, film, or similar types of media. Furthermore, a sound recording and playback

device, such as a speaker and/or microphone may optionally be coupled to bus 411 for audio interfacing with computer system 400. Another device that may be coupled to bus 411 is a wired/wireless communication capability 425 to communication to a phone or handheld palm device. Note that any or all of the components of system 400 and associated hardware may be used in various embodiments of the present invention. However, it can be appreciated that other configurations of the computer system may include some or all of the devices.

[0042] The foregoing discussion merely describes some exemplary embodiments of the present invention. One skilled in the art will readily recognize from such discussion, the accompanying drawings and the claims that various modifications can be made without departing from the spirit and scope of the appended claims. The description is thus to be regarded as illustrative instead of limiting.